UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

NETWORK APPLIANCE, INC., Plaintiff-Counterclaim Defendant, v. SUN MICROSYSTEMS, INC., Defendant-Counterclaim Plaintiff.	CASE NO. C-07-06053-EDL **DECLARATION OF DR. SCOTT BRANDT IN SUPPORT OF SUN MICROSYSTEMS, INC.'S OPENING CLAIM CONSTRUCTION BRIEF**

I, Scott Brandt, Ph.D., declare as follows:

I. INTRODUCTION

1. I have been retained by Sun Microsystems as an expert witness with regard to certain patents being asserted in Case No. 6053, including US. Patent Nos. 5,819,292 ("the '292 patent") and 6,892,211 ("the '211 patent").

2. I am over the age of eighteen and I am a citizen of the United States.

3. Attached hereto as Exhibit 1 is a true and correct copy of my current *curriculum vitae* ("CV").

4. I received a Bachelor of Mathematics from the University of Minnesota, Minneapolis in 1987, and a Master of Science in Computer Science, also from the University of Minnesota, in 1993. In 1999, I received a Ph.D. in Computer Science from the University of Colorado at Boulder.

5. I am presently Professor of Computer Science at the University of California Santa Cruz ("UCSC"). I am also Director of the UCSC/Los Alamos Institute for Scalable Scientific Data Management Director (ISSDM), Director of the UCSC Systems Research Laboratory (SRL), and co-founder of the UCSC Storage Systems Research Center (SSRC). My research is in the area of Computer Systems, specializing in both Storage Systems and Real-Time Systems. I joined UCSC as an Assistant Professor in 1999, was promoted to Associate Professor (with tenure) in 2003, and was promoted to Professor in 2007.

6. Before joining UCSC, I spent a number of years doing research and development in the computer industry.

7. While I was working toward my Doctorate in Computer Science in 1994–1998, I worked as Instructor, Research Assistant and Teaching Assistant in the Computer Science Department of the University of Colorado, Boulder.

8. From 1992–1994, I worked as a Senior Computer Scientist at Secure Computing Corporation, in Roseville, Minnesota. My responsibilities included researching secure operating systems, maintaining the storage subsystem of Secure Computing's operating system, and co-developing Secure Computing's next generation operating system.

9. From 1991–1992, I worked as a Senior Research Scientist for Alliant Techsystems Research and Technology Center (Formerly Honeywell Systems and Research Center)("Alliant") in Hopkins, Minnesota. My responsibilities at Alliant included researching and developing real-time imaging processing systems.

10. In 1990, I co-founded Theseus Research ("Theseus") in Minneapolis, Minnesota, and served as its Vice President from 1990–1991. Theseus was a small company devoted to researching asynchronous circuit technology and parallel computer languages.

11. From 1987–1990, I worked as a Senior Research Scientist at Honeywell Systems and Research Center (SRC) in Minneapolis, Minnesota. My responsibilities at Honeywell included researching and developing real-time image processing systems.

12. I have secured over $10,000,000 in research funding from the National Science Foundation, the National Laboratories, the Department of Energy Office of Science, the

Department of Education, and private industry for projects related to computers science, file systems, storage systems and other related technologies. A detailed list of my research funding is found in my CV.

13. I am a member of the following professional societies: the Institute of Electrical and Electronics Engineers ("IEEE") (Senior Member), the USENIX association, and the Association of Computing Machinery ("ACM"). I have served as General Chair, Program Chair, or Program Committee Member for numerous academic conferences and have served as a research grant proposal reviewer for the National Science Foundation and the Science Foundation Ireland. A detailed list of my other professional activities, memberships and speaking engagements is found on my CV.

14. As of this writing, I have published over 100 scholarly articles and research papers, many in the area of file and storage systems, including 12 journal articles, 53 conference papers, and 37 workshop or short papers. All journal articles, book chapters, and other publications I have authored are listed in my CV.

15. I am a named co-inventor on seven U.S. patents, listed in my CV.

16. I have considered the '292 and '211 patents for the purposes of providing my expert opinion regarding the meaning of certain claim terms. The '292 and '211 patents cover aspects of storage technology with which I am greatly familiar, due to my background, training and experience. The '292 and '211 patents relate to storage systems, and more particularly, to ways of maintaining a consistent file system in storage systems and for creating read-only copies of the file system. (See e.g., '292 patent at col. 1, lines 14–16.) Because of my background, training and experience, I am qualified as an expert in the technical areas relevant to these patents.

17. In this declaration, after describing the materials I reviewed, I will first describe some aspects of file system technology that will assist in understanding the invention described in the '292 and '211 patents. Next, I will review technology pertinent to the '292 and '211 patents. I will then identify various terms and phrases used in the claims of the '292 and '211 patents and provide an explanation of how one of ordinary skill in the art of computers, computer systems and

architecture would have understood the various claim terms and phrases in these claims at the time of invention.

II. MATERIALS REVIEWED

18. To prepare this declaration, I reviewed each of the ‘292 and ‘211 patents, prosecution histories of each of the ‘292 and ‘211 patents including references cited therein, U.S. Patent No. 7,174,352 (the “’352 patent”), which is related to the ‘211 and ‘292 patents through a series of continuation and continuation-in-part applications; the prosecution history of the ’352 Patent; and the definitions of certain terms as found in technical dictionaries. I have also reviewed the definitions for the terms proposed by Sun and NetApp.

19. Relevant excerpts of the technical dictionaries I reviewed are attached as Exhibit 2 (IEEE Standard Dictionary of Electrical and Electronics Terms, 6th Ed. (1996)), Exhibit 3 (Webster’s Dictionary of Computer Terms, Eighth Ed. (2000)), and Exhibit 4 (Dictionary of Storage Networking Terminology, Common Storage Networking-Related Terms and their Definitions, by the Storage Networking Industry Association (SNIA) (http://www.snia.org/education/dictionary/c/#copy_on_write)).

III. BACKGROUND OF FILE SYSTEM TECHNOLOGY

A. File Systems

20. File systems are organized structures for short- or long-term data storage in computer systems together with the software for accessing those structures and storing and retrieving data[1]. Fundamentally, a file system is an electronic version of a file cabinet—a collection of data (*e.g.* files) and management structures (*e.g.*, directories, inodes and other file system structures).

21. There are many different file systems with different characteristics that depend upon the goals of the file system developers. File systems may be designed for personal data storage, corporate data storage, multimedia data storage, database data storage, *etc*. Examples of file systems include:

---

[1] The term “file system” is commonly used to refer to the structures, the software that implements and accesses them, or both.

- The FAT file system, developed by Microsoft as part of the DOS system;
- The UNIX Fast File System (FFS), developed by McKusick *et al*. as part of the Berkeley Unix system;
- The NTFS file system, developed by Microsoft as part of the Windows operating system(s);
- XFS, a high-performance file system developed by Silicon Graphics;
- OFC, (Optical File Cabinet), a copy-on-write file system originally implemented on optical disks;
- ZFS, a scalable file system developed by Sun Microsystems as part of their Solaris operating system; and
- WAFL, a file system developed by Network Appliance (NetApp) for their file server appliance products.

22. Because file systems are often designed to provide a common, compatible service, conforming to a standard interface specification, such as the POSIX File System standard or the NFS network file system protocol, many file systems provide very similar services implemented using very similar structures. Common features include files for storing data, directories for organizing the files, inodes for bookkeeping associated with each file, organizing the information about specific files, a free block map for maintaining the information about the storage that is not (currently) part of any file or file system structure, and a root file system structure stored at a known location on the disk to allow the computer to locate the other structures each time the system starts up, *e.g.*, after a reboot.

23. While serving a common purpose in very similar ways, file systems are often designed by different people, to execute as part of different operating systems (*e.g.*, Windows, Unix, *etc*.), and with slightly different goals, and so may vary in their implementation details. For example, file systems designed for very large storage servers may use more efficient data structures for their directories and free block maps than those designed for use in personal computers. File systems may also be implemented locally, for example to execute on a PC that stores data on its hard drive, or remotely, as part of a storage server or appliance. Remote file

systems are generally accessed over a network using a network file system protocol such as NFS or CIFS.

B. Consistent and Inconsistent States

24. Consistency is a basic concept in data storage. The structures of a file system may be in consistent or inconsistent states depending upon whether or not they agree with respect to the status of the data or other structures they refer to.

25. The inode of a file typically refers to the blocks of disk storage that contain the data for the file, while the free block map refers to the blocks of storage that are not part of any file. Inodes and block maps are file system structures containing information that describes a file system's contents and state. Information in the file system structures is used by the file system to locate and retrieve data, such as files that have been stored in the file system by its users. File system structures are also used by the file system for internal bookkeeping, such as keeping track of used and unused disk blocks.

26. An inconsistent state may occur when, for example, an inode of a file refers to a block that is also referred to by the free block map. Another inconsistent state may occur if, for example, the inodes for two different files refer to the same data block. This particular inconsistent state may be arrived at due to an occurrence of the previous inconsistent state as the result of the apparently (but not actually) free block being allocated to a second file. This inconsistent state may cause several problems; for example, data stored in the first file may inadvertently be overwritten by data written to the second file. A less dangerous inconsistent state occurs when a data block is referred to nowhere, neither in any inode nor in the free block map.

27. The corruption or absence of even one block within a file system (for example, of an inode block holding a pointer to a disk block storing file data) may leave the file system in an inconsistent state because the file system is not able to correctly find and operate upon all of its data.

28. File systems strive to maintain consistency at all times, but may fail to do so due to programming errors or, commonly, due to computer system crashes at inopportune moments. For example, if a file needs to grow to store more data, it may need additional blocks. Blocks are

typically added to a file by 1) scanning the free block map to discover blocks that are currently free on the disk, 2) removing references to them from (or otherwise marking them "used" in) the free block map, and 3) adding references to them in the file's inode structure. If steps 2 and 3 are done in the opposite order and the system crashes in between the two steps, then the next time the file system starts up it will be in an inconsistent state, with a block referred to by both a file inode structure and the free block map. By contrast, a crash between steps 2 and 3 (in their correct order) will result in a block not being referred to anywhere, which is a harmless inconsistency.

29. A well-known method for avoiding file system inconsistencies is to carefully order the operations such that a crash at any time will not result in any harmful inconsistencies. To achieve this, file systems like LFS, OFC, and WAFL move the file system from one consistent state to another "atomically"—by updating the root structure so that it roots the new consistent state only after all operations since the last consistent state have been successfully committed to disk. In these file systems, file system blocks are not updated in place, *i.e.*, modified data is not written over the old data. Instead, as explained in more detail below, these file systems use variations on a *copy-on-write*[2] scheme to write all new and changed data to unallocated blocks. As a result, the old blocks comprising the previous consistent file system state are not overwritten and remain available should the file system crash before the new root structure is written.

C. Snapshots

30. Many modern file systems provide the ability to produce *snapshots* of the file system. A snapshot is a usable copy of a file system (or some sub-set of data in the file system) as it appeared at the time that the snapshot was initiated. Snapshots are used, for example, for backing up the file system.

31. Snapshots may be efficiently implemented via copy-on-write techniques, in which file system data and structures are duplicated (and modified) as they are changed, with the

---

[2] "A technique for maintaining a point in time copy of a collection of data by copying only data which is modified after the instant of replicate initiation. The original source data is used to satisfy read requests for both the source data itself and for the unmodified portion of the point in time copy." Ex. 4, A Dictionary of Storage Networking Terminology, Common Storage Networking-Related Terms and their Definitions, by the Storage Networking Industry Association (SNIA) (http://www.snia.org/education/dictionary/c/#copy_on_write)

snapshot referring to the old copies and the active file system referring to the new copies. Both the snapshot and the active file system refer to all file system structures and data that are unchanged. At the time a snapshot is initiated, all data and most structures should be the same, and so copy-on-write allows for very efficient initiation of snapshots. Subsequent operations may be more expensive as data structures must be duplicated as the file system contents are modified to allow the active file system to reflect the changes without changing the snapshot.

D. The ‘292 and ‘211 patents

32. The ‘292 and ‘211 patents refer to features of NetApp’s WAFL file system. WAFL was designed by NetApp for their network appliance file server (filer) products. NetApp’s filers are remote network-accessible storage devices that provide storage service to a large number of storage *clients*, which are computer systems that access the shared storage service provided by the filer.

33. The specific features of WAFL that are disclosed and claimed are its structures and mechanisms for maintaining the on-disk structures of WAFL in consistent states and for creating snapshots.

34. WAFL operates by using copy-on-write techniques for nearly all of the file system. Any time data or file system structures are changed in the WAFL file system, they are written to new locations on the disk. Modified data and structures (*e.g.* inodes) are buffered in memory until a new consistency point is to be written, at which time they are written to disk in new locations. The last thing to be updated is the root inode, which is stored in a fixed location (or locations) on the disk and is used to locate all of the other structures in the file system. By copying all other modified structures and data blocks first and updating the root inode last, WAFL ensures that the on-disk structures are always consistent (although they may not always refer to the most recently updated data). Snapshots are accomplished by copying the root inode, which refers to the structures making up the current consistent state of the file system, to a new location.

IV. BACKGROUND OF WAFL PATENTS

A. Introduction

35. The original specification of the ’292 and ’211 patents was filed as part of U.S. Application No. 71,643 on June 3, 1993. This application was continued as U.S. App. No. 454,921, which matured into the ’292 patent. The ’211 patent, filed on April 12, 2004, is a continuation of the original ’643 application. Both these patents (and many other patents that incorporate the ’292 patent by reference) were developed during the availability of WAFL. The patents related from the ’643 application are therefore referred to as the WAFL patents

B. Prior File Systems

36. As discussed in Section II above, most file systems strive to maintain consistency and a number of prior file systems used techniques similar to those employed by WAFL and documented in the ’292 and ’211 patents.

37. Like WAFL, the Episode File System, published by Chuntani *et al*. in 1992, supports the creation of a duplicate of the file system, or “clone”, via copy-on-write (COW) techniques. Also similar to WAFL, all data and metadata in Episode are accessible through a table of inodes, called anodes. (These and some other aspects of the operation of Episode are discussed in the “Background Art” section of the ’292 patent at Col. 1:35–3:35.)

38. The Berkeley Fast File System (FFS), was developed by McKusick *et al*. and published in 1984. FFS was a reimplementation of the Unix File System from Bell Laboratories and was designed primarily to provide higher performance than was possible with the original Unix File System. FFS uses a number of now-common concepts including replicated super-blocks, large blocks, cylinder groups, bitmapped free block lists, clustering, and a program called a file system consistency checker (fsck).

39. FFS uses several techniques to maintain file system consistency, including write ordering. Write ordering writes critical information to the hard drive in an order that minimizes later problems should the computer crash between writes.

40. Like WAFL, other file systems have been based on extensive use of copy-on-write. For example, the Log-structured File System (LFS), first conceived in 1988, is very similar

to WAFL. Like WAFL, LFS uses copy-on-write for all file system data and metadata except for a root data structure at a known location. As in WAFL, all updates to the LFS file system are written into free space on the disk. Like WAFL, LFS moves from consistent state to consistent state upon the completion of each write to disk by updating a root structure.

C. Basic On Disk Data Structures in the WAFL Patents

41. Like many file systems, WAFL is designed to store file data on hard disk drives ("hard drives" or "disks"). '292 col. 5:48–51. Most hard drives store data in 512-byte units called sectors, which are the smallest amount of data that may be read or written at one time. By design, hard drives retain information when the power is turned off, allowing a computer to store and later retrieve important information, including, for example, the stored programs that the computer runs as well as data such as digital photos, email, documents, or financial records.

42. Sectors on a disk are numbered. They are accessed by specifying the number of the sector to be read or written. Historically, these numbers were statically assigned to individual disk sectors, with each sector receiving a unique number. This presented some problems when specific sectors on the disk were found to be corrupted, *i.e.*, incapable of reliably storing information, as it required each file system to discover and maintain a record of the bad sectors. Modern hard drives instead use logical sector numbers that may or may not correspond to the physical sector numbers. Importantly, bad sectors can be trivially "mapped out", or avoided, by not being assigned a logical sector number or by having their number reassigned to a spare (good) sector elsewhere on the disk.

43. Disk sectors typically contain 512 bytes of data. If managed and accessed individually, this relatively small amount of data per sector can cause significant overhead in the file system. Thus, it is common to group sectors into "blocks", or groups of 2, 4, 8, or more, so that each data access instead accesses a block of sectors, reducing the per-byte overhead of data access. '292 patent col. 5:49–51. Like sectors, blocks are often referred to by number. In WAFL, 4KB blocks of data without fragments are used for both for both the on-disk and in-core (in-memory) data storage. '292 patent col. 5:49–50 and col. 6:58–60.

44. Data in WAFL (and other file systems) is organized into files, which may be

composed of zero or more data blocks. Each file is described by a data structure called an inode. The inode contains the information required to locate and access the file data. Each inode may refer to up to 16 other blocks that may contain data or further references to other blocks. '292 patent col. 5:48–64.

45. Files in WAFL (and other file systems) are contained in directories. '292 patent col. 5:49–53. Directories are special files that contain the names of other files and identifiers indicating which files the names refer to (often by inode number). Files are located by looking in the appropriate directory and finding the identifier that corresponds to the file's name. Because directories are themselves files, directories also have inodes and are listed in other directories. '292 patent col. 5:49–53 and col. 8:57–col. 9:17.

46. All WAFL inodes, except that of the inode file itself, are stored in a special meta-data file called the inode file. The inode file contains a list of inodes, each in-use inode referring to a single file. '292 patent col. 5:56–57 and col. 9:26–32.

47. The root inode, the inode for the inode file itself, is stored in a fixed location on the disk in a structure called the fsinfo data structure. The fsinfo data structure is unique to WAFL and allows a system to locate the inode file and begin accessing the file system. '292 patent col. 9:32–36 and col. 10:57–col. 11:5.

48. The inode map, or inomap, is another special meta-data file that contains information used in the operation of the WAFL file system. The inode map is a file indicating which of the inodes in the inode file are in use. When a new file is created, the inode map is searched to find a free inode to use to describe the file. The inode map is then updated to indicate that the selected inode is in use. '292 patent col. 10:18–56.

49. The block map is another special meta-data file used by the WAFL file system. The block map serves a similar purpose to the inode map, indicating whether each of the blocks on disk is available or in use by the current file system or a snapshot. When data is to be written to disk, the block map is searched to locate a free block, one not used by the current file system or any snapshot, in which to write the data, then updated to reflect that the block is no longer available. '292 patent col. 9:49–col. 10:18.

D. Basic In Memory Data Structures the WAFL Patents

50. In normal operation, the contents of a file system change as files are created, deleted, and updated. For performance reasons, such changes may be held in memory for a short while before being written to disk. A typical file system will write data to disk after it has sat in memory for too long (a configurable parameter) or when it must be written to free up memory for other data.

51. In file systems like FFS, data is updated on a file-by-file basis. That is, file data is updated and file metadata is updated accordingly, independent of updates to other files. The file data that has changed may overwrite the old contents of the file by reusing the same disk blocks. The updated metadata may similarly overwrite the old metadata. In WAFL, all of the data and metadata changes that have occurred recently are committed to disk together, to free blocks on disk. This includes all changes to the inode file, inode map, and block map. Once all of the updated data and metadata has been written to disk, the fsinfo structure is updated to refer to the newly updated inode file. ‘292 patent col. 11:60–12:14.

52. Between updates to the disk all changed data (and metadata) are buffered in memory. ‘292 patent col. 12:40–48. WAFL uses in-memory data structures that are similar to the on-disk data structures described above: in-memory data is stored in 4 KB memory blocks referenced by in-memory inode structures. The in-memory data structures differ slightly from the corresponding on-disk structures; for example, the in-memory inodes specify whether the in-memory data is the same or different from that on disk. ‘292 patent col. 6:53–8:56.

E. Indirect Blocks

53. WAFL inodes and corresponding data structures are similar to those found in other file systems. Each WAFL inode contains information about the file and 64 bytes that may contain file data (if the file is small enough) or 16 direct or indirect references to disk blocks containing

the file data. '292 patent col. 5:62–6:13 and illustrated in Fig. 4B.

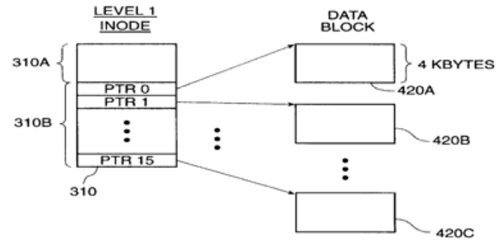

FIG. 4B

54. The inode structure is scalable to store large data files by using multiple layers of indirection. If the file data is 64 bytes or less it is included directly in the inode. If the file data is between 64 bytes and 64 KB, then the inode contains up to 16 pointers that point directly to 4 KB data blocks containing the file data. If the data file is larger than 64 KB, then scalability is achieved by having each of the inode block pointers point to an "indirect" block. Depending upon the size of the file, each indirect block then either points to a data block or another indirect block. *Id.*, col. 6:18–53, Figs. 4B–4D. This is illustrated in the '292 patent in Figure 4D, which shows three levels of indirection:

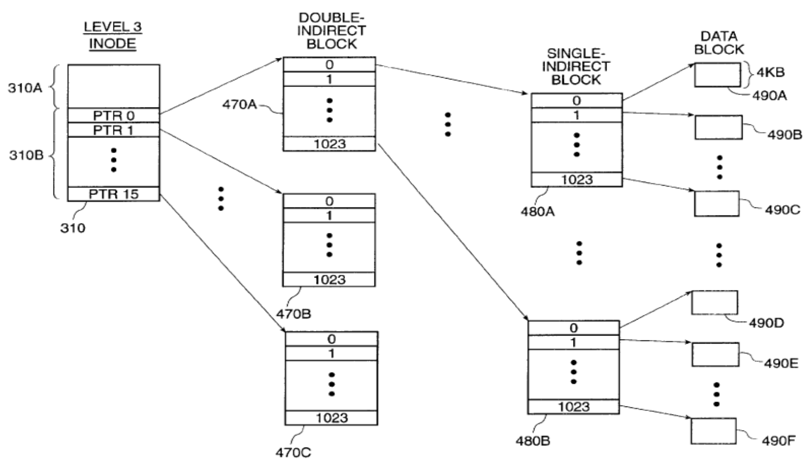

FIG. 4D

55. In Figure 4D, each of the 16 pointers in the "Level 3 Inode" points to a "Double Indirect Block." Each of the Double Indirect blocks have 1024 pointers that in turn point to 1024 "Single Indirect Blocks." Each Single Indirect Block also has 1024 pointers, each of which

points to a 4 KB data block. Thus, up to 64 GB of data blocks can be addressed. *Id.*, col. 6:35–52, Fig. 4D.

F. Moving To A New Consistent State.

56. When all of the 4 KB data blocks and associated metadata files and the root inode are written to disk and are identifiable to the file system, the file system is consistent. ’292 patent, col. 11:6–col. 12:1 and illustrated in Figure 16. A feature of the ’292 patent is that the file system written to the disks remains consistent while modifications are concurrently being made to the data and written to disk. ’292 patent, col. 11:65–66. Between consistent states, the modified blocks (described as dirty blocks) are accumulated and organized in the in-memory buffer blocks, as discussed above. ’292 patent, col. 7:17–27, col. 8:8–56.

57. To create a consistency point, WAFL always writes new or modified data to unallocated blocks on disk so that it never overwrites existing data. ’292 patent, col. 2–4. To form a new consistency point, all of the modified blocks other than the fsinfo block, are first written to disk. ’292 patent, col. 14:24–27. The patent requires that the fsinfo block be written last because it contains the root inode that roots the file system. ’292 patent, col. 10:66–64, col. 12:9–11, col. 14:24–27. “A new consistency point is occurs when the fsinfo block is updated by writing a new root inode …” ’292 patent, col. 4:16–18.

58. If the system crashes, the system is able to locate each of two copies of the fsinfo block due to their being stored in fixed locations. ’292 patent, col. 10:58–60, col. 13:65–67. With the root inode of the fsinfo structure, the file system can identify all other blocks using the pointer from the root inode to the inode file, which in turn contains the “inodes describing all other files in the file system.” ’292 patent, col. 9:6–27, col. 10:62–66. By triggering the change to the new consistency point upon writing the fsinfo, the file system progresses to a new consistency point atomically and retains tight control over the file system. ’292 patent, col. 12:25–26, col. 14:26–28.

59. The file system described within the ’292 patent provides additional capability to retain a copy of past consistent states in a read-only form, called a snapshot. ’292 patent, col. 4:20–21, col. 17:66–col. 18:3. A snapshot is similar to a past consistency point that has been

recorded. ’292 patent, col. 20: 21–22. The presence of a snapshot ensures that blocks associated with the snapshot will not be overwritten. ’292 patent, col. 4:33–35. Thus, snapshots can be used to backup file systems while continuing to allow access to the file system or to restore data that has been lost in the active file system. ’292 patent, col. 3:66–col. 4:3.

60. The ’292 patent emphasizes the value of being able to maintain multiple snapshots. ’292 patent, col. 3:66– col. 4:3, col 20:28–30. “WAFL supports up to 20 different snapshots that are numbered 1 through 20.” ’292 patent, col. 18:8–9. The ability to record multiple snapshots is enabled by providing numerous bits for each block in a block map table. “The key data structures for snapshots are the blkmap entries where each entry has multiple bits for a snapshot. This enables a plurality of snapshots to be created.” ’292 patent, col. 20:15–18. By setting a bit (equal to 1) in a block’s entry in the block map, the file system ensures that the block will not be overwritten and that it will remain part of a snapshot. ’292 patent, col. 18:26–30. By providing a block map with multiple available bits for snapshots on a per block basis, the ’292 patent provides storage to record numerous snapshots of consistency points of the file system.

G. Disputed Claim Terms.

1. “non-volatile storage means”

61. The term “nonvolatile storage means” appears in asserted claims 4 and 8 of the ‘292 patent. I am told that the use of the word “means” in the claim raises a presumption that the term is a means-plus-function limitation. I am also told that this presumption can be rebutted if the claim includes sufficient structure to perform the function in its entirety. In my opinion, neither claim 4 nor claim 8 includes sufficient structure to perform the stated function in its entirety.

62. Claims 4 and 8 use the term “non-volatile storage means” in order to describe how data blocks of a file system are stored. Claim 4 says “a method for maintaining a file system comprising blocks of data stored in blocks of a non-volatile storage means at successive consistency points …” ‘292 patent, col. 25:12–14. Therefore, in my opinion, a function of the non-volatile storage means is “storing data blocks of a file system.” Because the storage is “non-volatile,” the data blocks must be maintained even when there is no power. [Ex. 3, Webster’s

Dictionary of Computer Terms, Eighth Ed. at 376.] Therefore, the function of the non-volatile storage means is, in my opinion, "storing blocks of data for a file system so that the data is maintained in the absence of power."

63. Claim 8 says "a method for creating a plurality of read-only copies of a file system stored in blocks of a non-volatile storage means…" '292 patent, col. 26:1–3. As discussed in the technology background section of my declaration, a file system like WAFL is comprised of many data blocks and the claim specifically refers to storing data in "blocks of a non-volatile storage means…" Therefore, storing read-only copies of a file system inherently includes storing data blocks. In my opinion, the stated function for the term "non-volatile storage means" in claim 8 is the same as claim 4, namely, "storing blocks of data for a file system so that the data is maintained in the absence of power."

64. In my opinion, the claims do not include any structure for storing blocks of data for a file system so that the data is not lost in the absence of power. The only remotely "structural" language is the "non-volatile storage means" itself. But the term "non-volatile storage" does not identify any specific structure to one of ordinary skill in the art. The word "non-volatile" only suggests a way of keeping data so that it is not lost without power and does not refer to any specific structure for doing so. Ex. 3, Webster's Dictionary of Computer Terms, Eighth Ed. at 376. The term "storage" is defined as "[t]he act of storing information" or "any device in which information can be stored, sometimes called a memory device." Ex. 2, IEEE Standard Dictionary of Electrical and Electronics Terms, 6th Ed. at pp. 1049. Therefore, the word "storage" does not suggest any specific physical structure, device, or class of devices.

65. Even when the words are used together in the phrase "non-volatile storage," one of ordinary skill in the art would not understand the phrase to refer to or define any particular physical structure or class of devices for performing the recited function. There are many unique and completely unrelated structures that can be used to retain information in a way that does not require power. For example, paper, tape, punch cards, film, hard disks, battery-backed RAM devices, write-once optical disks, ROM devices, flash memory, and countless other unrelated types of devices can be used to store information in the absence of electrical power. Therefore, in

my opinion, the term “non-volatile storage means” does not identify a particular physical structure or class of physical structures.

66. I am told that in order to interpret a means-plus-function term, one must identify the corresponding structure in the specification that performs the function. As I stated above, the function of the “non-volatile storage means" is “storing blocks of data for a file system so that the data is not lost in the absence of power.” In my opinion, one of ordinary skill in the art would understand that the structure disclosed for performing this function is “one or more disks with a block-based format (*i.e.*, 4 KB blocks that have no fragments), where the disk storage blocks are the same size as the data blocks of the file system.”

67. The file system described in the ‘292 patent specification is referred to as a “Write Anywhere File-system Layout (WAFL).” ’292 patent, col. 5:48–50. The specification only describes the invention in the context of the structures and operation of the WAFL system. ’292 patent, col. 5:45–col. 6:52 (describing “On-Disk WAFL Inodes”), col. 6:53–8:56 (describing “In-core WAFL Inodes”), col. 8:57–col. 9:17 (describing WAFL directories), col. 9:18–col. 11–27 (describing WAFL meta-data files), col. 11:28–58 (describing WAFL inodes having “dirty” blocks), col. 11:62–col. 17:63 (discussing consistency points in a WAFL system), col. 17:64–col. 24:6 (discussing snapshots in a WAFL system). As I mentioned in the technology background section, fundamental to the operation of WAFL is “a disk format system that is block based (*i.e.*, 4 KB blocks that have no fragments),” and the use of “inodes” to describe its files. ’292 patent, col. 5:48–53. The block-based disk format system disclosed in the ’292 patent is the only structure disclosed for storing data blocks of a file system so that information is not lost in the absence of power. In the block-based disk format system, the blocks on disk, which are 4 KB in size with no fragments, are designed to be the precisely the same size as the 4 KB data blocks that make up the files of the file system. ’292 patent, col. 6:23–53 and Figs. 4B–4D. Therefore, the only structure identified by the specification for storing data blocks of a file system is “one or more disks with a block-based format (*i.e.*, 4 KB blocks that have no fragments), where the disk storage blocks are the same size as the data blocks of the file system.”

2. "meta-data for successive states of said file system"

68. The term "meta-data for successive states of said file system" appears in claim 8 of the '292 patent. In my opinion, one of ordinary skill in the art would understand that the term "meta-data for successive states of said file system" in claim 8 means: "a block map file for recording snapshots of the file system."

69. Claim 8 goes beyond the formation of "consistent states" of a file system, which are discussed in claim 4, and further requires recording a plurality of these consistent states as "read-only copies." '292 patent, col. 8:1. These read-only copies of the file system are referred to as "snapshots" by the '292 patent. '292 patent, col. 4:21–22. The necessary step of "marking said blocks of said non-volatile storage means…as comprising a respective read-only copy of said file system," (*Id.*, col. 26:12–15) ensuring blocks of a snapshot are not lost by being overwritten, is accomplished by setting the appropriate bits within the block map file. *Id.*, col. 4:35–43. The claim language describes this necessary step as storing the "meta-data for successive states of said file system." *Id.*, col. 26:6.

70. The claim language supports Sun's construction. To create a plurality of snapshots, claim 8 sets forth multiple steps for identifying blocks that are members of any snapshot. Specifically, the claim language says that "meta-data" is stored and copied to identify blocks used within the file system and its snapshots. '292 patent, col. 26:4–12. This requirement of the claim means that each block must be identified within the meta-data and that the meta-data must be stored for successive snapshots. *Id.*, col. 26:6–10. Therefore, the claim language lays out a map where each block has entries for membership in successive snapshots.

71. The specification also supports Sun's construction. The specification says the file system "uses files to store meta-data that describes the layout of the file system," and that allocated blocks are identified within the blockmap (blkmap) file. '292 patent, col. 5:53–58; col. 9:54–55. The invention of the '292 patent uses the block map file to track block within successive states of a file system. *Id.*, col. 4:36–40.

72. The '292 patent describes the block map file in great detail. The block map assigns a 32-bit entry to each block in the disk system. '292 patent, col. 9:51–53. The block map

file contains meta-data associated with snapshots, as shown in Figure 21E:

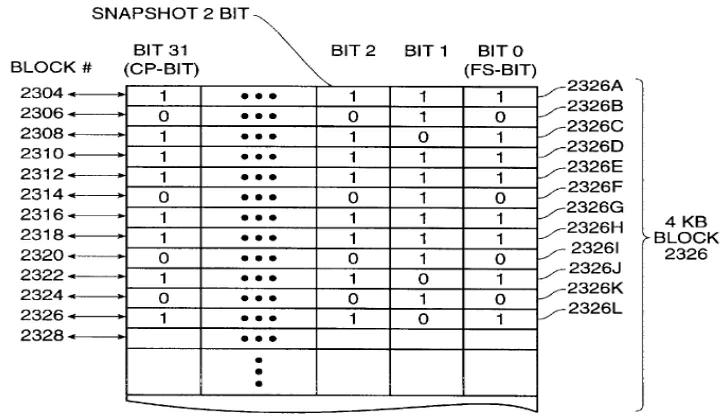

FIG. 21E

73. Each block has a corresponding entry in the block map file and is given a block number, shown above in the range 2304 through 2328. *Id.*, col. 9:51–53, col. 16:22–23. The first 21 bits (bits 0–20) identify whether a block is part of the current state or of any snapshots of the file system. *Id.*, col. 9:59–65. The value stored at bit 0 indicates whether the block is part of the active file system. *Id.*, patent, col. 10:7–9. Bit 1 is the first snapshot bit (corresponding to snapshot number 1), Bit 2 is the second snapshot bit (corresponding to snapshot number 2), and so on. *Id.*, col. 16:29–20, col. 23:2–5. A "1" is written to bits 0–20 to indicate whether the block is part of the active file system and/or any snapshots. For a block to be "free", all entries within the block map file for that block must contain a "0" entry. *Id.*, col. 9:66 – col. 10:1.

74. The '292 specification describes only one method for marking blocks as compromising snapshots, namely, via a block map. *Id.*, col. 4: 39–43.

75. In my opinion, NetApp's construction is too broad and contradicts the teachings of the specification. First, NetApp's construction eliminates the need for a block map, which is required by the '292 patent specification. Also, NetApp replaces the word "meta-data" with the broader term "information." The '292 patent specification carefully defines only certain specific files as being "meta-data," namely, i) an inode file, ii) a inomap file; and iii) a blkmap file. *Id.* at col. 9:21–22. At the same time, the specification states that other "information" pertaining to a file system is <u>not</u> "meta-data." For example, the '292 patent teaches that "WAFL uses files to

store meta-data" (*Id*. at col. 5:53–54) and the "[f]ile system information structure 1510 is not a meta-data file but is part of the WAFL system." *Id.* at col. 10:60–62. Therefore, NetApp's construction, which uses the broader term "information" in place of "meta-data," contradicts the specification, which says that not all information pertaining to a file system is "meta-data".

76. The prosecution history does not suggest that the meta-data used to track the state of blocks is accomplished by anything other than the block map file. Thus, in my opinion, a person of ordinary skill in the art would understand that the term "meta-data for successive states of said file system" means a "block map file for recording snapshots of the file system."

3. "file system information structure"

77. The term "file system information structure" appears in asserted claim 4, as well as in non-asserted claims 2, 5 and 7. In my opinion, one of ordinary skill in the art would understand that a "file system information structure" means: a "data structure that contains the root inode of a file system in a fixed location on disk." This is Sun's proposed construction.

78. The '292 patent specification supports Sun's proposed construction. The specification equates the claim term "file system information structure" with the "fsinfo" block and uses those terms interchangeably. The specification also defines these two equivalent terms to be a "data structure at a fixed location on disk that contains the root inode of the file system." '292 patent, col. 9:33–36, col. 10:57–60.

79. The term "file system information structure" is clearly defined in the '292 patent specification. The specification repeatedly equates the file system information structure with the "fsinfo" block. First, the specification of the '292 patent repeatedly places the parenthetical definition "(fsinfo)" next to or within the phrase "file system information structure." For example, the specification twice refers to the "file system information (fsinfo) structure" and twice cites the "file system information (fsinfo) block." '292 patent, col. 9:33–36, col. 10:57–65, col. 13:63–64.

80. The specification also uses the terms "file system information structure" and "fsinfo block" and "fsinfo structure" interchangeably. Col. 11:3–4, col. 12:27–37. For example, the specification says "FIG. 15 is a diagram illustrating a file system information (fsinfo)

structure 1510." *Id.*, col. 10:57–58. Figure 15 labels structure 1510 "fsinfo block." *Id.*, Fig. 15. This is seen throughout the specification. The "fsinfo block" serves the same function in Figures 16, 18A–18C, 20B, 21A–21D, 21F, 22 and 23A, and in the corresponding text of the specification relating to those figures. *Id.*, col. 11:6–27, col. 18:50–col. 19:2. The '292 patent is consistent in defining the "file system information structure" by using it synonymously with, the "fsinfo block."

81. The specification is also very clear in explaining the necessary requirements of the file system information (fsinfo) structure. As I discussed in paragraphs 47 and 56–58 of my declaration, a fundamental aspect of the invention(s) described in the '292 patent is the structure, function and location of the "root inode." The specification is very clear that the invention requires the "file system information structure" to be kept in a fixed location and include the root inode. See for example, '292 patent, col. 9:33–35; col. 10:57–64; and Figs. 20A–20C, 21A–21D, 21F (each showing the "fsinfo block" as including the "root inode").

82. The phrase "file system information structure" is not commonly used within the storage industry, and does not have an established meaning within the industry. Based upon the language of claim 4 and the clear meaning that the specification gives to this term, a person of ordinary skill in the art would understand that the "file system information structure" refers to the fsinfo structure, which must include the root inode and must be stored at a fixed location on disk.

83. The specification also explains that the file system information structure is used for a fundamental stated purpose of the invention: to "progress[] from one self-consistent state to another self-consistent state." '292 patent, col. 4:11–12, col. 4:16–18, col. 12:16–20. The patent says that "[a] new consistency point occurs when the fsinfo block is updated by writing a new root inode …." *Id.*, col. 4:16–18, col. 12:16–20. Thus, a person of ordinary skill in the art knows the file system information structure cannot change consistency points without the presence of the root inode in that structure.

84. In my opinion, the fixed location of the fsinfo block is also necessary to the invention described in claim 4 ("maintaining a file system … at successive consistency points"). The ability to recover the file system after a power interruption is only possible if the root inode

can be located. ’292 patent, col. 14:22–29, col. 13:66–col. 14:3. This is made possible by placing the root inode within the fsinfo structure and storing the fsinfo structure at a fixed location on disk. Col. 9:34–35, col. 10:57–65, col. 11:3–5, col. 14:22–20. Thus, the ‘292 patent teaches that in order to achieve the stated purpose of claim 4, the file system information structure must include the root inode and must be at a fixed location on disk. ’292 patent, col. 10:58–60.

85. In my opinion, NetApp’s construction is too broad because it appears to capture any kind of inode disclosed in the specification—not just root inodes, and does not require it to be in a known, fixed location on disk. I find no support in the specification for NetApp’s construction.

86. In my opinion, the specification is very careful and intentional in its use of the term “file system information structure” and its root inode contents that are stored in a fixed location on disk. Therefore, in my opinion, the term "file system information structure" means “data structure that contains the root inode of a file system in a fixed location on disk.”

V. U.S. PATENT NO. 6,892,211

A. Technology Background.

87. The ’211 patent is a continuation of the ’292 patent. The ’211 patent claims are concerned with the process of moving the file system from one consistent state to another. That process was claimed at a high level in claim 4 of the ’292 patent, considered above. The ’211 patent claims a first consistent file system state stored on disk and rooted by an on-disk root inode. The ’211 patent also claims a second consistent file system state. ’211 patent, col. 23:64–67. The second consistent state has not yet been committed to disk, so in the ’211 patent the second consistent state is “in transit” from the in-core buffers to new blocks on disk. *Id*., col. 24:1–11. Thus, the ’211 patent claims are akin to a photographic snapshot of the dynamic process of moving the WAFL file system from one consistent state to another.

88. I have already discussed many of the file system structures referenced in the ’211 patent claims above in connection with the ’292 patent, so the discussion below may be somewhat repetitive. However, it is worth discussing some of these structures and concepts again

in the particular setting of the '211 patent.

89. In the '211 patent file system, files are stored as 4 KB blocks on disk. Each file is described by its inode, which holds information describing the file, such as ownership, file size and access time, and pointers to the 4KB blocks storing file data. '211 patent, col. 5:50–54, 6:4–7. Each pointer is essentially a 4-byte long block number. *Id.*, col. 5:63–65. When the file system accesses an inode, it uses these block numbers to figure out which blocks store the file data. Thus, block numbers stored in an inode are said to "point" to storage blocks. An inode contains space for 16 pointers referring directly or indirectly to data blocks comprising the file's data. *Id.*, col. 6:20–24. For a very small file of 64 bytes or less, the space for the pointers is instead used for the data itself. For a file larger than 64 bytes and smaller than 64 KB, the inode directly points to its data blocks (16 blocks of 4 KB making up the 64 KB maximum capacity), as shown in Fig. 4:

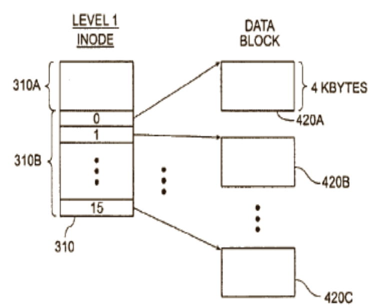

FIG. 4B

90. For files bigger than 64 KB and smaller than 64 MB, the inode points to a set of indirect blocks, which in turn point to data blocks, as shown in Fig. 4C:

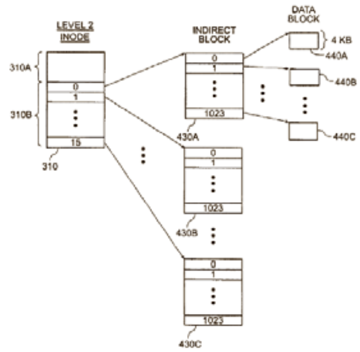

FIG. 4C

91. The inode in Fig. 4C directly references (*i.e.* points to) indirect blocks 430A–C and indirectly references data blocks 440A–C. *Id*., col. 6:25–35. Indirect referencing means that the inode itself does not store the block number pointers for blocks storing file data, instead an intermediate data structure, an indirect block (or blocks), are used to store those pointers. In this scenario, the inode only stores pointers to the intermediate blocks. Additional levels of indirection allow the inode to serve as the root of a progressively larger file. *Id*., col. 6:36–53. The '211 patent specification notes that WAFL inodes are distinct from prior art inodes because they only support one level of indirection, meaning that a WAFL inode never points to indirect blocks of different levels. *Id*., col. 5:63–67. As I noted earlier, however, the inode structure of WAFL is similar to that of prior art file systems like FFS and LFS. All of these file systems use inode structures to root a multi-level tree where data blocks are the outermost "leaves" and intermediate blocks are the "branches." In my opinion, the distinctions are trivial and, in any case, are unclaimed.

92. WAFL always writes file data blocks, inodes and all other file system structures (with the critical exception of the fsinfo structure, which contains the root inode) to unallocated blocks on disk, wherever such blocks are available. '211 patent, col. 4:18–21, 5:50–54. Before exploring the important consequences of this arrangement, it is worth discussing how the '211 patent file system would find a file. After all, users do not reference files by their inode numbers. Instead, users know files by file names. When a user wants to find a file, the file system needs to match the file name with the number of the inode identifying that file. This is accomplished by

looking in the directory containing the file and looking up the inode number associated with the name of the file listed in that directory.

93. In practice, file systems typically contain many directories, many of which are inside other directories. File systems, including WAFL, find a specific directory by first consulting a root directory, or base directory that directly or indirectly contains all other files and directories in the file system. In the ’211 patent, the root directory is shown as structure 1650 in Fig. 16:

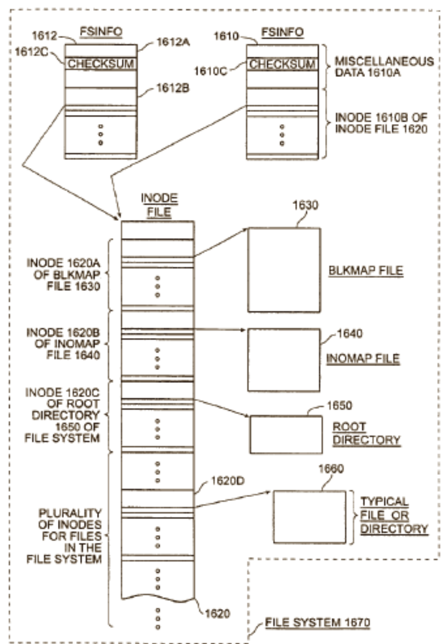

FIG. 16

94. The WAFL file system accesses the root directory by finding its inode in the inode file 1620. (The inode file is a special metadata file that stores all of the file inodes, except for its own. *Id.*, col. 9:19–32. I discuss it in greater detail below.) It finds the inode file by first reading the root inode stored at a known location in the fsinfo block. (The root inode points to the inode file. *Id.*, col. 9:32–35.) Once the root directory has been located, any other file or directory can be located by looking up the next level directory in the current directory and proceeding recursively through the levels of the hierarchy until the desired file or directory is located.

95. Once the inode number of a file (or directory) is ascertained, the file system reads the inode file (inomap) to obtain the actual inode. (In practice, most of these structures are stored in in-core buffers, so that the file system does not have to read from disk every time it needs to

read or modify them. Reading from in-core buffers is significantly faster than reading from disk.) The file system would locate the inode within the inode file using the inode number it looked up in the directory. That inode, in turn, contains pointers to the data blocks comprising the file data. This method of file lookup is almost universal in the file system world and is done in a similar way by prior art file systems like FFS and LFS.

96. As can be seen, even a basic operation like looking up a file requires that the file system be able to locate several inodes (inodes for the root directory, intermediate directories, and the actual file), which are stored in a structure called the inode file. This, of course, means that the inode file must always be quickly and easily locatable. If it is not, the file system would be unusable and inconsistent—it would not be able to locate any of its files.

97. As the specification acknowledges at col. 9:25–29, prior art file systems like FFS had an inode table—a structure storing inodes, like the ’211 patent file system inode file. In FFS, this structure was at a fixed place on disk. In the file system of the ’211 patent the inode file is written anywhere on disk. *Id.*, col. 9:25–29. This is also true of prior art *copy-on-write* file systems like LFS. In such file systems, a root structure at a fixed, known location points to the ever-moving inode file (inode table).

98. In the file system of the ‘211 patent, this root structure is called the root inode. *Id.*, col. 4:15–18, 9:32–35. It contains pointers that directly point to a special metadata file, the inode file, that contains the inodes of all of the other files in the file system, as shown in Fig. 17B:

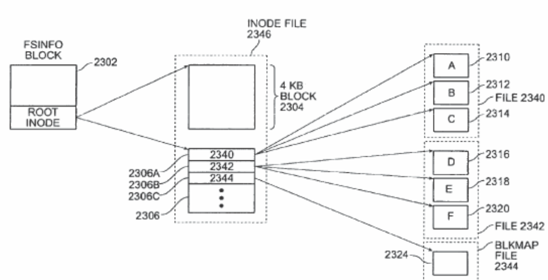

FIG. 17B

99. In Fig. 17B, the root inode is located in the fsinfo block 2302 and points to blocks

of the inode file 2346, which contain file inodes that point to files 2340, 2342 and BLKMAP (block map) file 2344, one of the special metadata files I discuss in greater detail below. *Id*., col. 15:55–16:3. Thus, the root inode points directly to the blocks of the inode file and indirectly to all of the other blocks in the file system. *Id*., Fig. 16, col. 11:20–25. Altogether, the blocks pointed to by the root inode comprise a file system in a consistent state on disk, as claimed in the following ’211 claim 1 language:

> maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system. *Id*., col. 23:64–68.

100. The structure I just described is static. The file system, however, is a dynamic entity, with files constantly being changed, added and deleted. To accomplish this, file systems bring the on-disk data and structures into computer memory. For example, sections of inode tables (or, in the case of the ’211 patent, the inode file) may be cached in memory so that the file system does not have to access the disk every time it looks up an inode. Data blocks of files undergoing modifications are also stored in memory and modified there, before being written back out to disk.

101. In order to make modifications to data stored in a consistent state on disk, the ’211 patent file system caches a copy of the file inodes and their data blocks in “in-core” (in-memory) buffers. *Id.*, col. 6:55–7:17. Each in-core inode contains the information of the corresponding on-disk inode, including its block pointers, as well as some additional data, including pointers to in-core buffers holding modified file data. *Id*., *see also* ’211 patent, Fig. 10, col. 7:6–17. Like other inodes, the on-disk root inode stored in its fsinfo block is also copied to an in-core root inode. ’211 patent, col. 17:35–57. The in-core cache thus serves the ’211 patent file system and prior art file systems as a “holding pen” for data undergoing changes before such data is written back to permanent storage, such as a disk.

B. Disputed Claim Terms To Be Construed.

1. "pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system"

102. This term appears in all of the independent claims—claims 1, 9 and 17. I have considered this term in the context of the overall claim language and the specification and it is my opinion that the plain meaning of "pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system" is "pointing directly and indirectly to buffers in said memory and pointing directly and indirectly to a second set of blocks on said storage system." This is the only construction of this term that is consistent with the claim language and with the workings of the disclosed invention.

103. In evaluating the claim language, I considered that the preceding limitation requires that the on-disk root inode "point[] directly and directly to a first set of blocks" that store a first consistent file system state. '211 patent, col. 23:64–68. As I explained above, this refers to the on-disk root inode pointing directly to the inode file and, through it, pointing indirectly to all other parts of the file system. *Id.*, col. 9:25–35, Fig. 16, col. 11:6–27.

104. When the on-disk structures like the inode file and root inode are brought in-core, their relationship remains unchanged—the in-core root inode points to the in-core inode file, which points to all other files. *Id.*, col. 9:35–49, Fig. 17B, col. 15:24–41, 15:56–16:3. Because these structures are in-core, they are stored in buffers – areas of memory allocated to hold data normally resident in disk blocks. *Id*. Therefore, just like the on-disk root inode points directly ***and*** indirectly to blocks comprising a first consistent state, its in-core cousin, the in-core root inode, points directly ***and*** indirectly to buffers comprising a later, second, consistent state. The second consistent state represents the evolution of the active file system that occurs in-core until it is fully committed to disk by updating the on-disk root inode with its in-core state. At that time, the "second" consistent state in the claim becomes the "first" on-disk consistent state. This progression from one consistent state to another was discussed earlier in connection with the '292 patent.

105. I further considered the purpose of the buffers and blocks referenced by the claims.

The claims language teaches that these buffers and blocks pointed to by the in-core root inode serve a very specific purpose—together they "stor[e] data and metadata for a second consistent state of said file system." '211 patent, col. 24:4–6. Thus, the in-core root inode is assigned a unique task—to root a file system in a consistent state. Furthermore, as I explain below, the "metadata" referenced in the claim language is understood by one of ordinary skill to comprise at least three critical metadata files, without which the file system of the '211 patent would not be able to store a consistent state. The specification teaches that to store a consistent state one of these files is always pointed to by the root inode directly and two others are always pointed to indirectly, thereby confirming Sun's proposed claim construction.

106. The files in question are the inode file, the blockmap file and the inode map file. The specification teaches that "WAFL keeps information that describes the file system in files known as meta-data. Meta-data comprises an inode file, inomap file, and a blkmap file." '211 patent, col. 9:19–24. Thus, the specification teaches that these are structural files that "describe" the file system. Such files are critical to file system operation and, therefore, are necessary for keeping the file system in a consistent state.

107. I described the inode file at some length above—it is the only file to which the root inode points directly: "[t]he inode file 1210 is pointed to by an inode referred to as the 'root inode'." '211 patent, col. 9:32–33. The inode map file contains the inodes of all other files in the file system, including the block map and inode map files. *Id.*, col. 9:25–32, col. 11:6–27. Because of this, the file system would be inoperable without this file.

108. The block map file keeps track of which blocks are allocated in the current file system, which blocks are used by any of the snapshots, and which blocks are free for the file system to write to. *Id.*, col. 9:50–65. This meta-data file is critical to keeping the file system in a consistent state. Without the block map, the '211 patent file system would not know which blocks are included in the current (first) or previous consistent states and would not be able to allocate blocks for the second claimed consistent state. Thus, the very operation claimed in the '211 patents—transitioning to a second consistent state—would not be achievable without this meta-data file.

109. The third critical meta-data file is the inode map file. ’211 patent, col. 10:20–49. This file is used by the file system to keep track of which inodes in the inode file are unallocated. *Id*.

110. As described by the specification, these three files represent the minimum necessary structures (in addition to the root inode) that are necessary to constitute a file system in a consistent state. Therefore, the claimed meta-data must comprise these files. (Claim 8 of the ‘292 patent referenced one of these specific files, i.e., the block map file, which is used to store successive states of the file system.)

111. The specification teaches that the root inode points to the inode map directly and to the other two meta-data files indirectly: “[b]ecause the root inode 1610B and 1612B of the fsinfo blocks 1610 and 1612 describes the inode file 1620, that in turn describes the rest of the files 1630–1660 in the file system including all meta-data files 1630–1640, the root inode 1610B and 1612B is viewed as the root of a tree of blocks.”[3] *Id*., col. 11:20–27. Thus, in order for the root inode to serve its claimed function of rooting “buffers and said second set of blocks storing data and meta-data for a second consistent state of said file system,” the root inode must point directly to the inode map ***and*** point indirectly to the meta-data files (blockmap and inode map) as well as the regular data files.

112. In confirming my understanding of the construction of “pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system,” I also considered how the specification describes the claimed mechanism of transitioning from a first to a second consistent state.

113. As shown in Fig. 17B, the second consistent state is initially identical to the first—there have been no changes since the last consistent state was committed to disk. In Fig. 17C, changes have been made in the active file system stored in in-core buffers—data F in block 2320 of file 2342 has been changed to F′, while data C in block 2314 has been deleted from file 2340:

---

[3] The specification describes two root inodes. The ’211 file system uses two root inode copies for reliability. ’211 patent, col. 12:24-36.

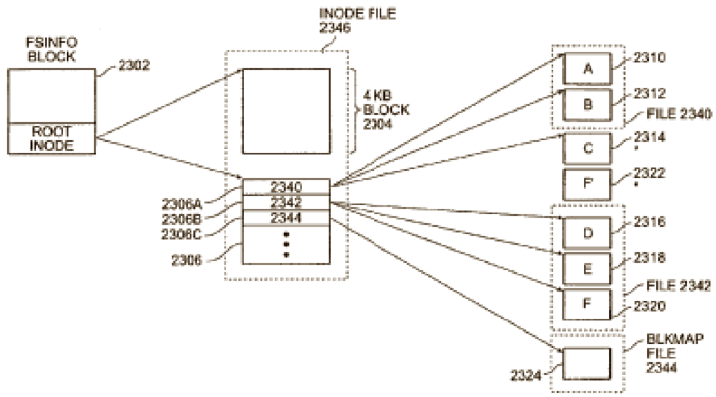

FIG. 17C

Thus, a buffer 2322 has been allocated in memory to store modified data F′. Blocks 2314 and 2322 are now marked with a single asterisk to indicate that they are "dirtied" – holding modified data. *Id.*, col. 16:4–15.

114. Changes in the file data, in turn, mean that block 2306 of the inode file containing inodes 2340 and 2342 for files 2340 and 2342 must be updated as it is now incorrect (in Fig. 17C it still points to block 2314 even though data C has been deleted and it does not point to block 2322 holding data F′). To do this, a new buffer 2308 holding inodes 2340 and 2342 is allocated in memory and updates to these inodes are recorded, as shown in Fig. 17E:

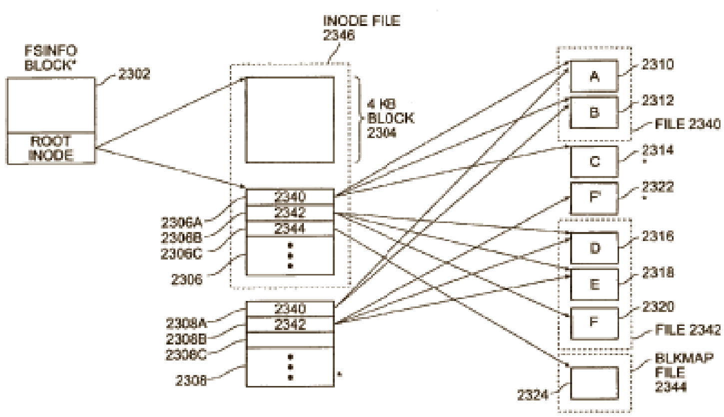

FIG. 17E

115. Fig. 17E shows that updated inodes 2340 and 2342 stored in block 2308 now point to the updated file 2340 (comprising unchanged blocks 2310 and 2312, but not block 2314, which was deleted) and 2342 (comprising unchanged blocks 2316 and 2318 and new block 2322, but

not block 2320, which still holds old data F). ’211 patent, col. 16:36–16:67. Note that inode file block 2306 still points to the old state of its data. This old state (the claimed first consistent state) still exists on disk because the root inode has not been updated. *Id.*, Fig. 17F, col. 16:64–67.

116. At this point, blocks containing changes to regular files are flushed (written out) to disk. *Id.*, col. 16:48–49. Thus, buffer 2322 containing changed block with data F′ in file 2342 is allocated disk space and it is written out to disk. *Id.*, Fig. 17F, col. 16:63–67. In the next step, the modified block map file 2344 is written out to disk—this meta-data file changes whenever there are any changes in the file system. *Id.*, col. 17:1–3.

117. As shown in Fig. 17L, the in-core root inode is now updated to point directly and indirectly to buffers and point directly and indirectly to a second set of blocks that together comprise the second consistent state:

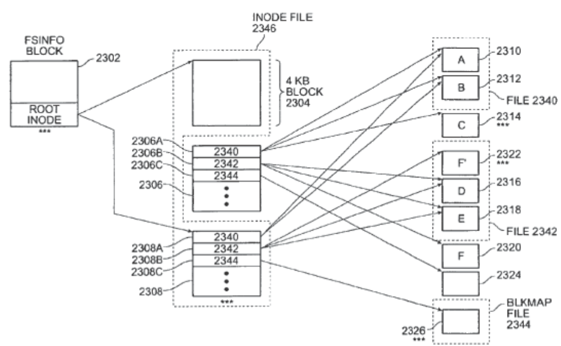

FIG. 17L

118. This figure shows the transitory file system state that is claimed in the ’211 patent. The in-core root inode stored in the fsinfo block 2302 now points directly to the inode file buffer 2308 holding inodes for modified files 2340, 2342 and the blockmap file 2344; the in-core root inode simultaneously points indirectly to buffers holding modified data for these files. *Id.* As discussed above, these buffers are flushed to disk in stages, so that the in-core root inode directly and indirectly points to a new set of blocks comprising the changed data and inodes. ’211 patent, col. 17:35–57. At the same time, the in-core root inode still points directly to unchanged blocks of the inode file (*e.g.* block 2304 in Fig. 17L) and indirectly to the corresponding unchanged data blocks (not shown in Fig. 17L).

119. As can be seen, the changes between the first and second consistent state are stored in buffers and newly written blocks, which are not pointed to by the old root inode because they

are not pointed to by the old inode file. This corresponds to the claim language requiring that "changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set f blocks not pointed to by said on-disk inode." *Id.*, col. 24:8–11.

120. The description above is a summary of the relevant operational features of the '211 patent file system disclosed in the specification. This description unambiguously discloses that in order to root a consistent state, the in-core root inode always points directly and indirectly to buffers and points directly and indirectly to blocks.

121. The specification discloses only this one embodiment. There is no teaching or even suggestion that the claimed file system may be able to root a consistent state without directly and indirectly referencing the critical meta-data files discussed above.

122. I further considered some description in the specification (col. 7:55–57) that teaches that regular inodes point to "indirect and/or direct blocks." This statement in the specification bears no relevance to the proper understanding of the claim term at issue because it refers to regular inodes. Regular inodes do not root a file system in a consistent state, therefore the specification does not require them to reference special meta-data files.

123. I have considered NetApp's proposed construction, which I understand to be that "pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system" means "pointing directly to blocks and/or buffers, and/or indirectly to blocks and/or buffers." I note the obvious dissonance of NetApp's proposed construction with the plain claim language. I further note that this construction, for example, may be read as simply *e.g.* "pointing directly to blocks." The claim language, however, requires that "said buffers ***and*** said second set of blocks stor[e] data and meta-data for a second consistent state of said file system." '211 patent, col. 24:4–6 (emphasis added). These buffers and second set of blocks must be pointed to by the root inode for the claim to make any sense, but this is not required by NetApp's construction. For these reasons and for all the reasons I discussed above, I do not believe that one of skill in the art would understand the plain claim language to have the meaning ascribed to it by NetApp.

2. "root inode"

124. The term "root inode" appears in independent claims 1, 9 and 17. Sun's proposed construction is that "root inode" means "the index node data structure stored in a fixed location that roots a set of self-consistent blocks on the storage system that comprise the file system." I have considered this claim term in the context of the claim language and the specification and I believe that Sun's proposed construction is correct.

125. As I discussed above in the technology background section, a file system needs a root structure to access its other structures. In order to make a practical working file system, it is well understood in the art that the root structure should be stored in a known location. The root then provides a fixed starting point that allows the file system to locate any of its blocks without prior knowledge of their location.

126. A file system state wherein the system lacks knowledge of location of some or all of its data is common: it occurs any time the power has been turned off for any reason. Because the most up-to-date file system state is typically located in-core—in a memory that is often (though not always) of a type that loses its contents when power is interrupted—crashes, power failures, or simply turning off the power can wipe out the system's knowledge of the file system's data structures. In that case, the file system needs to boot from its root—a fixed, known structure—and re-discover the rest of its data and meta-data by traversing the structures referenced to by the root.

127. This necessity for a fixed place from which the file system can boot is satisfied in the '211 patent by the claimed root inode—the inode that logically sits at the top of the file system and is located in a fixed place. The specification teaches at col. 10:59–61 that "[t]he root inode 1510B of a file system is kept in a fixed location on disk so that it can be located during booting of the file system." In the file system of the '211 patent this fixed location is the location of the fsinfo block: "[t]he root inode is kept in a fixed location on disk referred to as the file system information (fsinfo) block described below." '211 patent, col. 9:33–35. Two copies of the fsinfo structure containing the root inode are stored in fixed locations on disk for reliability.

Id., col. 11:3–5. To transition from one consistent state to another, the '211 patent teaches that the root inode is flushed from the in-core cache to these two fixed locations on disk. *Id.*, col. 12:1–23.

128. In fact, it is easy to see that in the '211 patent file system the root inode ***must*** be at a fixed location. As discussed above, the claimed root inode is the inode that locates the inode file. *Id*., col. 9:32–33. The inode file stores inodes for all of the other files in the file system. *Id*. It is one of the critical meta-data files that, as I discuss above, comprise the "meta-data" that the claims say is stored in buffers and blocks of a second consistent state. *Id*, col. 24:2–6. If volatile memory is used to store these buffers and power is lost, the data in the buffers will be lost.

129. To recover, the file system must regain access to the on-disk inode file. However, because of the "write-anywhere" nature of the claimed file system, blocks of the inode file are written to any available locations on disk, shifting around as data is modified and the file system progresses through consistent state cycles. *Id*., col. 9:19–24. The only practical way to find the location of the inode file in such a file system is to store a pointer to it in a location that never changes. This is exactly the solution taught by the '211 patent, which discloses that the on-disk root inode is stored in a known location within the fsinfo block. *Id*., col. 9:25–35.

130. For all of these reasons, a construction of the term "root inode" has to address the fundamental problem that the patent solves through this structure—of being able to always locate a constantly moving data structure that is part of the claimed file system in a consistent state. To do so, the patent teaches only one solution—to store the root inode at a fixed, known place. No other solution is taught by the patent or would be apparent to one of ordinary skill in the art. Therefore, one of ordinary skill in the art would understand the root inode to be stored in a fixed location, consistent with the claims, the specification and the knowledge and experience of one of ordinary skill.

3. "consistent state"/"state of a file system"

131. The term "consistent state" appears in independent claims 1, 9 and 17. The term "state[s] of a file system" appears in claim 8 of the '292 patent. I understand that the parties have agreed that these terms should have the same construction. Sun's proposed construction is that

"consistent state" means "a set of storage blocks for that file system that includes all blocks required for the data and structure of the file system." I have considered this claim term in the context of the claim language and the specification and I believe that Sun's proposed construction is correct. In evaluating the proper construction for this term, I further considered NetApp's '352 patent, which at col. 4:25–31 provides a definition of consistency that is fully in line with Sun's proposed construction.

132. Because file system consistency is such a fundamental concept, I have addressed it in the Background of File System Technology section.

I declare under the penalty of perjury under the laws of the United States of America that the foregoing is true and correct. This declaration is executed on this July 8, 2008, at Palo Alto, California.

Scott Brandt

DR. SCOTT BRANDT